```
1 #pragma once
 2 #include <Windows.h>
 3
 4 class AudioManager
 5 {
 6
        static AudioManager* s_pInstance;
 7
 8
        bool m_SoundOn;
 9
10
        AudioManager()
11
            : m_SoundOn(true)
12
13
14
        }
15
16
        //is this a singleton design pattern? one and only one instance?
        // global access, no ownership, lazyinitialisation
17
18
        // saves memory - but how?
        // Flexibility
19
20
21 public:
        static AudioManager* GetInstance()
22
23
24
            if (s_pInstance == nullptr)
25
            {
26
                s_pInstance = new AudioManager();
27
28
            return s_pInstance;
29
        }
30
31
        static void destroyInstance()
32
        {
33
            delete s_pInstance;
34
            s_pInstance = nullptr;
35
        }
36
37
        void ToggleSound()
38
        {
39
            m_SoundOn = !m_SoundOn;
40
        }
41
42
        bool IsSoundOn()
43
        {
44
            return m_SoundOn;
45
        }
46
47
        void playdoorclose()
48
            if (!m_SoundOn)
49
50
            {
51
                return;
52
            }
            Beep(500, 75); // frequency and duration
53
```

```
54
             Beep(500, 75);
 55
         }
 56
 57
         void playerdooropen()
 58
 59
             if (!m_SoundOn)
 60
 61
                  return;
 62
 63
             Beep(1397, 97);
 64
         }
 65
 66
         void pickupkey()
 67
             if (!m_SoundOn)
 68
 69
             {
 70
                  return;
 71
 72
             Beep(1568, 100);
 73
         }
 74
 75
         void dropKeySound()
 76
 77
             if (!m_SoundOn)
 78
             {
 79
                  return;
 80
 81
             Beep(1568, 200);
 82
             Beep(1568, 50);
 83
         }
 84
 85
         void moneySound()
 86
             if (!m_SoundOn)
 87
 88
 89
                  return;
90
 91
             Beep(1568, 50);
 92
         }
 93
 94
         void loseLife()
 95
 96
             if (!m_SoundOn)
 97
 98
                  return;
 99
             Beep(200, 100);
100
101
         }
102
103
         void PlayLoseSound()
104
105
             if (!m_SoundOn)
106
             {
```

```
107
                 return;
108
             }
109
             Beep(500, 75);
             Beep(500, 75);
110
             Beep(500, 75);
111
112
             Beep(500, 75);
113
             Beep(500, 75);
             Beep(500, 75);
114
115
         }
116
117
         void win()
118
119
             if (!m_SoundOn)
120
             {
121
                 return;
122
             }
             Beep(1568, 200);
123
124
             Beep(1568, 200);
             Beep(1568, 200);
125
             Beep(1245, 1000);
126
             Beep(1397, 200);
127
128
             Beep(1397, 200);
             Beep(1397, 200);
129
130
             Beep(1175, 1000);
131
         }
132 };
```

```
...and Polymorphism\Game\Game Inheritence\AudioManager.cpp
1 #include "AudioManager.h"
3 AudioManager* AudioManager::s_pInstance = nullptr;
4
5
```

```
1 #include "PlaceableActor.h"
2
3 class Door : public PlaceableActor
4 {
5 public:
 6
       Door(int x, int y, ActorColour colour, ActorColour closedColour);
       virtual void Draw() override;
7
8
       virtual ActorType GetType() override { return ActorType::Door; }
9
       bool IsOpen() { return m_isOpen; }
10
11
       void Open() { m_isOpen = true; }
12
13 private:
14
       bool m_isOpen;
       ActorColour m_closedColour;
15
16
17 };
```

```
1 #include <iostream>
 2 #include <Windows.h>
 3 #include "Door.h"
 5 Door::Door(int x, int y, ActorColour colour, ActorColour closedColour)
       :PlaceableActor(x, y, colour)
 6
7
       , m_isOpen(false)
 8
       , m_closedColour(closedColour)
9 {};
10
11 void Door::Draw()
12 {
13
       HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
14
       if (m_isOpen)
15
           SetConsoleTextAttribute(console, (int)m_colour); // cast to an int
16
17
       }
18
       else
19
       {
           SetConsoleTextAttribute(console, (int)m_closedColour);
20
21
22
       std::cout << "|";
23
       SetConsoleTextAttribute(console, (int)ActorColour::Regular);
24 }
```

```
1 #include "PlaceableActor.h"
 2
 3 class Enemy : public PlaceableActor
4 {
 5 public:
 6
       Enemy(int x, int y, int deltaX = 0, int deltaY = 0);
7
       virtual ActorType GetType() override { return ActorType::Enemy; }
 8
9
       virtual void Draw() override;
       virtual void Update() override;
10
11
12 private:
13
14
       int m_movementInX;
       int m_movementinY;
15
16
17
       int m_currentMovementX;
18
       int m_currentMovementY;
19
20
       int m_directionX;
       int m_directionY;
21
22
23
       void updateDirection(int& current, int& direction, int& movement);
24
25 };
```

```
1 #include "Enemy.h"
 2 #include <iostream>
 3 #include <Windows.h>
 4
 5
   Enemy::Enemy(int x, int y, int deltaX, int deltaY)
        : PlaceableActor(x, y, ActorColour::Green) // placing initial coordinates →
 6
           of enemy
 7
        , m_currentMovementX(0)
        , m_currentMovementY(0)
 9
       , m_directionX(0)
10
        , m_directionY(0)
       , m_movementInX(deltaX) // The maximum distance the enemy can move in
11
          the x-direction
12
        , m movementinY(deltaY) // The maximum distance the enemy can move in
          the y-direction
13 {
14
       if (m_movementInX != 0)
15
       {
            m directionX = 1;
16
17
        }
       if (m_movementinY != 0)
18
19
20
            m_directionY = 1;
21
        }
22 }
23
24 void Enemy::Draw()
25 {
26
       HANDLE console = GetStdHandle(STD OUTPUT HANDLE);
       SetConsoleTextAttribute(console, (int)m_colour);
27
        std::cout << (char)153; // prints coloured enemy.</pre>
29
       SetConsoleTextAttribute(console, (int)ActorColour::Regular);
30
   }
31
32 void Enemy::Update() // update the state of the enemy
33 {
34
       if (m movementInX != 0)
35
        {
            updateDirection(m_currentMovementX, m_directionX, m_movementInX);
36
37
        }
       if (m_movementinY != 0)
38
39
       {
            updateDirection(m currentMovementY, m directionY, m movementinY);
40
41
        }
42
       this->SetXYPosition(m_pPosition->x + m_directionX, m_pPosition->y +
43
          m_directionY);
44
   }
45
   void Enemy::updateDirection(int& current, int& direction, int& movement) // >>
      responsible for handling the movement of the enemy
47 {
48
       current += direction;
```

```
...itance and Polymorphism\Game\Game Inheritence\Enemy.cpp
```

```
if (std::abs(current) > movement) // reverse movement. if we reach the
        end we want to loop back the other way.

{
        current = movement * direction;
        direction *= -1; // change direction
    }
}

// If the absolute value of the current movement becomes greater than the
    maximum allowed movement (movement), it means the enemy has reached the end
    of its allowed movement range
```

```
...heritance and Polymorphism\Game\Game Inheritence\Game.h
1 #pragma once
 2 #include "GameStateMachine.h"
 3
 4
 5 class Game
 6 {
 7 public:
 8
       GameStateMachine* m_pStateMachine;
 9
10 public:
11
       Game();
       void Initialise(GameStateMachine* pStateMachine);
12
13
       void RunGameLoop();
       void Deinitialise();
14
15
16 private:
       bool Update(bool processInput = true);
17
       void Draw();
19
20 };
21
```

```
1 #include "Game.h"
 2
 3 Game::Game()
       : m_pStateMachine(nullptr)
 4
 5 {};
 7 void Game::Initialise(GameStateMachine* pStateMachine)
 8 {
9
       if (pStateMachine)
10
       {
           pStateMachine->Init();
11
12
           m_pStateMachine = pStateMachine;
13
       }
14 };
15 void Game::RunGameLoop()
17
       bool isGameOver = false;
       while (!isGameOver)
19
20
           Update(false);
21
           Draw();
22
           isGameOver = Update();
23
24
       Draw();
25 };
26
27 void Game::Deinitialise()
28 {
29
       if (m_pStateMachine)
30
           m pStateMachine->CleanUp();
32
33 };
34
35 bool Game::Update(bool processInput)
36 {
       return m_pStateMachine->UpdateCurrentState(processInput);
37
38 }
39
40 void Game::Draw()
41 {
42
       m_pStateMachine->DrawCurrentState();
43 }
```

```
1 #pragma once
2 #include "GameState.h"
3 #include "Player.h"
4 #include "Level.h"
 5 #include <Windows.h>
 6 #include <vector>
 7 #include <string>
 8
9 class StateMachineExampleGame;
10
11 class GameplayState :
       public GameState
12
13 {
14
       StateMachineExampleGame* m pOwner;
15
       Player m_player;
16
17
       Level* m_pLevel;
18
19
       bool m_beatLevel;
20
       int m skipFrameCount;
       static constexpr int kFramesToSkip = 2;
21
22
23
       int m_currentLevel;
24
       vector<string> m_LevelNames;
25
26 public:
       GameplayState(StateMachineExampleGame* pOwner);
27
28
       ~GameplayState(); // clean up after levelnames
       virtual void Enter() override;
29
       virtual bool Update(bool processInput = true) override;
30
31
       virtual void Draw() override;
32
33 private:
       bool load();
34
35
       void HandleCollision(int newPlayerX, int newPlayerY);
       void DrawHUD(const HANDLE& console);
36
37
38 };
39
40
```

```
1 #include "GameplayState.h"
 2
 3 #include <conio.h>
4 #include <iostream>
 5 #include <assert.h>
6
7 #include "Enemy.h"
8 #include "Key.h"
9 #include "Door.h"
10 #include "Money.h"
11 #include "Goal.h"
12 #include "AudioManager.h"
#include "Game.h"
14 #include "Utility.h"
15
16 #include "StateMachineExampleGame.h"
17
18 using namespace std;
19
20 constexpr int kArrowInput = 224;
21 constexpr int kLeftArrow = 75;
22 constexpr int kRightArrow = 77;
23 constexpr int kUpArrow = 72;
24 constexpr int kDownArrow = 80;
25 constexpr int kEscapeKey = 27;
26 constexpr int kBackspace = 8;
27
28 GameplayState::GameplayState(StateMachineExampleGame* pOwner)
29
       : m pOwner(pOwner)
30
       , m_beatLevel(false)
31
       , m skipFrameCount(0)
32
       , m currentLevel(0)
33
       , m_pLevel(nullptr)
34 {
35
       m_LevelNames.push_back("Level4.txt");
       m_LevelNames.push_back("Level5.txt");
36
       m_LevelNames.push_back("Level6.txt");
37
38 }
39
40 GameplayState::~GameplayState()
41 {
42
       m_pLevel = nullptr;
43
       delete m_pLevel;
44 }
45
46 bool GameplayState::load()
47 {
48
       if (m_pLevel)
49
50
           delete m_pLevel;
51
           m pLevel = nullptr;
52
       }
53
```

```
...nd Polymorphism\Game\Game Inheritence\GameplayState.cpp
```

```
2
```

```
54
         m pLevel = new Level();
 55
         return m pLevel->LoadLevel(m LevelNames.at(m currentLevel),
 56
           m_player.GetXPositionPointer(), m_player.GetYPositionPointer());
 57
    }
 58
 59 void GameplayState::Enter()
 60 {
         load();
 61
   }
 62
 63
 64 bool GameplayState::Update(bool processInput)
 65 {
         if (processInput && !m beatLevel)
 66
 67
             int input = _getch();
 68
 69
             int arrowInput = 0;
 70
             int newPlayerX = m_player.GetXPosition();
             int newPlayerY = m_player.GetYPosition();
 71
 72
             // One of the Arrow keys were pressed
 73
             if (input == kArrowInput)
 74
 75
             {
 76
                 arrowInput = _getch();
 77
             }
 78
 79
             if ((input == kArrowInput && arrowInput == kRightArrow) ||
                 ((char)input == 'd' || (char)input == 'D'))
 80
 81
             {
 82
                 newPlayerX++;
 83
             }
 84
             else if ((input == kArrowInput && arrowInput == kLeftArrow) ||
 85
                 ((char)input == 'a' || (char)input == 'A'))
 86
 87
             {
 88
                 newPlayerX--;
 89
             }
 90
             else if ((input == kArrowInput && arrowInput == kUpArrow) ||
 91
 92
                 ((char)input == 'w' || (char)input == 'W'))
 93
             {
 94
                 newPlayerY--;
             }
 95
 96
 97
             else if ((input == kArrowInput && arrowInput == kDownArrow) ||
                 ((char)input == 's' || (char)input == 'S'))
 98
 99
             {
100
                 newPlayerY++;
101
             }
102
103
             else if (input == kEscapeKey)
104
105
                 m pOwner->LoadScene
                                                                                    7
```

```
\dots \verb| nd Polymorphism \\ | Game \\ | Inheritence \\ | Gameplay \\ | State.cpp
```

```
3
```

```
(StateMachineExampleGame::SceneName::MainMenu);
106
             else if ((char)input == 'Z' || (char)input == 'z')
107
108
109
                 m_player.DropKey();
                 AudioManager::GetInstance()->dropKeySound();
110
111
             //If position never changed
112
113
             if (newPlayerX == m_player.GetXPosition() && newPlayerY ==
114
               m_player.GetYPosition())
115
116
             }
117
118
             else
119
             {
120
                 HandleCollision(newPlayerX, newPlayerY);
121
             }
         }
122
123
124
         if (m_beatLevel)
125
126
             ++m_skipFrameCount;
127
             if (m_skipFrameCount > kFramesToSkip) // player transitions over to >
               X spot before sound.
128
             {
129
                 m_beatLevel = false;
130
                 m_skipFrameCount = 0;
131
132
                 ++m_currentLevel;
                 if (m_currentLevel == m_LevelNames.size())
133
134
                 {
                     Utility::WriteHighScore(m_player.GetMoney());
135
                     AudioManager::GetInstance()->win();
136
137
                     m_pOwner->LoadScene
                                                                                     P
                       (StateMachineExampleGame::SceneName::Win);
138
                 }
139
                 else
140
                 {
141
                     load();
                 }
142
143
             }
144
         }
145
         return false;
146 }
147
148 void GameplayState::HandleCollision(int newPlayerX, int newPlayerY) // more >
      parameters to help with if loop
149 {
         bool isGameDone = false;
150
151
         PlaceableActor* collidedActor = m pLevel->UpdateActors(newPlayerX,
           newPlayerY); // creates a placeable actor
152
         if (collidedActor != nullptr && collidedActor->IsActive())
```

```
...nd Polymorphism\Game\Game Inheritence\GameplayState.cpp
153
154
             switch (collidedActor->GetType())
155
156
             case ActorType::Enemy:
157
                 Enemy* collidedEnemy = dynamic_cast<Enemy*>(collidedActor); //
158
                   specifies the type/ thing we are trying to cast, in this case →
                   an enermy
159
                 assert(collidedEnemy);
                 AudioManager::GetInstance()->loseLife();
160
161
                 // if the pointer is valid, if statement works, if it is a key
                   none of the code will work
162
                 collidedEnemy->Remove(); // if a collision with an enemy occurs, →
                    the enermy is removed.
163
                 m_player.SetXYPosition(newPlayerX, newPlayerY); // players
                   position is set to new position
164
                 m_player.DecrementLives(); // decrmeent lives
                 if (m_player.GetLive() < 0) // if less than zero game is over.</pre>
165
166
167
                     AudioManager::GetInstance()->PlayLoseSound();
168
                     m pOwner->LoadScene
                                                                                    P
                       (StateMachineExampleGame::SceneName::Lose);
169
170
                 break;
171
             }
172
             case ActorType::Money:
173
             {
                 Money* collidedMoney = dynamic_cast<Money*>(collidedActor); // >
174
                   if collided with money
175
                 assert(collidedMoney);
                 AudioManager::GetInstance()->moneySound();
176
177
                 collidedMoney->Remove(); // remove the money
                 m_player.AddMoney(collidedMoney->GetWorth()); // add the money
178
                   and show the worth.
179
                 m_player.SetXYPosition(newPlayerX, newPlayerY);
180
                 break;
             }
181
182
             case ActorType::Key:
183
             {
184
                 Key* collidedKey = dynamic cast<Key*>(collidedActor); //
                   returning null if fails within dynamic casts.
185
                 assert(collidedKey);
186
                 if (!m_player.HasKey())
187
                     m_player.PickUpKey(collidedKey);
188
189
                     AudioManager::GetInstance()->pickupkey();
190
                     collidedKey->Remove();
191
                     m_player.SetXYPosition(newPlayerX, newPlayerY);
192
193
                 break;
194
             }
195
             case ActorType::Door:
196
             {
```

```
...nd Polymorphism\Game\Game Inheritence\GameplayState.cpp
```

```
197
                 Door* collidedDoor = dynamic cast<Door*>(collidedActor);
198
                 assert(collidedDoor);
199
                 if (!collidedDoor->IsOpen())
200
                 {
201
                     if (m_player.HasKey(collidedDoor->GetColour()))
202
                     {
203
                          collidedDoor->Open();
204
                          collidedDoor->Remove();
205
                         m_player.UseKey();
206
                         m_player.SetXYPosition(newPlayerX, newPlayerY);
207
                         AudioManager::GetInstance()->dropKeySound();
                     }
208
209
                     else
210
                     {
211
212
                     }
213
                 }
214
                 else
215
                 {
216
                     m_player.SetXYPosition(newPlayerX, newPlayerY); // player
                       goes through the door
217
                 }
218
                 break;
219
             }
220
             case ActorType::Goal:
221
222
                 Goal* collidedGoal = dynamic_cast<Goal*>(collidedActor);
223
                 assert(collidedGoal);
224
                 collidedGoal->Remove(); // removes actors
225
                 m_player.SetXYPosition(newPlayerX, newPlayerY);
226
                 m_beatLevel = true;
227
                 break;
228
             }
229
             }
230
         }
         else if (m_pLevel->IsSpace(newPlayerX, newPlayerY)) // no collision
231
232
             m player.SetXYPosition(newPlayerX, newPlayerY);
233
234
         }
235
         else if (m_pLevel->IsWall(newPlayerX, newPlayerY))
236
237
             // wall collision
238
         }
239
    }
240
241 void GameplayState::Draw()
242
243
         HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
244
         system("cls");
245
246
         m_pLevel->Draw();
247
248
         //Set cursor position for player
```

```
...nd Polymorphism\Game\Game Inheritence\GameplayState.cpp
```

```
6
```

```
249
         COORD actorCursorPosition;
250
         actorCursorPosition.X = m_player.GetXPosition();
251
         actorCursorPosition.Y = m player.GetYPosition();
252
         SetConsoleCursorPosition(console, actorCursorPosition);
253
         m_player.Draw();
254
255
256
         //Set cursor to end of level.
257
         COORD currentCursorPosition;
258
         actorCursorPosition.X = 0;
259
         actorCursorPosition.Y = m_pLevel->GetHeight();
         SetConsoleCursorPosition(console, actorCursorPosition);
260
261
         DrawHUD(console);
262
263 }
264
265 void GameplayState::DrawHUD(const HANDLE& console)
266 {
267
         cout << endl;</pre>
268
         // Top Border
269
270
         for (int i = 0; i < m_pLevel->GetWidth(); ++i)
271
272
             cout << Level::WAL;</pre>
273
         }
274
         cout << endl;</pre>
275
         // left border
276
277
278
         cout << Level::WAL;</pre>
279
         cout << " wasd - move " << Level::WAL << " z - drop key " << Level::WAL;</pre>
280
281
         cout << "$: " << m_player.GetMoney() << " " << Level::WAL;</pre>
         cout << "Lives: " << m_player.GetLive() << " " << Level::WAL;</pre>
282
283
         cout << "Key: ";</pre>
         if (m_player.HasKey())
284
285
286
             m player.GetKey()->Draw();
287
         }
288
         else
289
290
             cout << " ";
291
         }
292
293
         // right border
294
295
         CONSOLE_SCREEN_BUFFER_INFO csbi;
296
         GetConsoleScreenBufferInfo(console, &csbi);
297
298
         COORD pos;
299
         pos.X = m pLevel->GetWidth() - 1;
300
         pos.Y = csbi.dwCursorPosition.Y;
301
         SetConsoleCursorPosition(console, pos);
```

```
...nd Polymorphism\Game\Game Inheritence\GameplayState.cpp
302
303
         cout << Level::WAL;</pre>
304
         cout << endl;</pre>
305
         // Bottom Border
306
307
         for (int i = 0; i < m_pLevel->GetWidth(); ++i)
308
309
              cout << Level::WAL;</pre>
310
         }
311
         cout << endl;</pre>
312 }
```

```
...ance and Polymorphism\Game\Game Inheritence\GameState.h
1 #pragma once
 2 // abstract class
 3
 4 class GameState
 7 public:
       virtual ~GameState() = default;
       virtual void Enter() {};
 9
      virtual bool Update(bool processInput = true) { return false; }
10
11
      virtual void Draw() = 0;
      virtual void Exit() {};
12
13
14
15
16 };
```

```
2 // abstract that will create the GameStates from.
4 class GameState;
6 class GameStateMachine
7 {
8
9 public:
10
11
      virtual ~GameStateMachine() = default;
12
13
      virtual bool Init() = 0;
      virtual bool UpdateCurrentState(bool processInput = true) = 0;
14
      virtual void DrawCurrentState() = 0;
15
      virtual void ChangeState(GameState* pNewState) = 0;
      virtual void CleanUp() = 0;
17
18
19 };
```

```
...heritance and Polymorphism\Game\Game Inheritence\Goal.h
1 #include "PlaceableActor.h"
 2
 3 class Goal : public PlaceableActor
 4 {
 5 public:
        Goal(int x, int y);
 7
        virtual ActorType GetType() override { return ActorType::Goal; }
        virtual void Draw() override;
 9
10
11 };
```

```
1 #include <iostream>
2 #include "Goal.h"
3
4 Goal::Goal(int x, int y)
5 : PlaceableActor(x, y)
6 {
7
8 }
9
10 void Goal::Draw()
11 {
12   std::cout << "X";
13 }</pre>
```

```
1 #pragma once
2
3 #include "GameState.h"
4 #include <set>
6 class StateMachineExampleGame;
8
9 class HighScoreState :
       public GameState
10
11 {
12
       StateMachineExampleGame* m_pOwner;
13
       std::set<int> m_highscore;
14
15 public:
       HighScoreState(StateMachineExampleGame* pOwner);
16
       ~HighScoreState() = default;
17
       virtual bool Update(bool processInput = true) override;
19
       virtual void Draw() override;
20
21
22
23 };
24
25
```

```
1 #include "HighScoreState.h"
 2
 3 #include <iostream>
 4 #include <conio.h>
 6 #include "StateMachineExampleGame.h"
 7 #include "Utility.h"
 8
9 HighScoreState::HighScoreState(StateMachineExampleGame* pOwner)
10
       : m_pOwner(pOwner)
11 {
12
       m_highscore = Utility::WriteHighScore(0);
13 }
14
15 bool HighScoreState::Update(bool processInput)
17
       if (processInput)
18
19
           int input = _getch();
20
           m pOwner->LoadScene(StateMachineExampleGame::SceneName::MainMenu);
21
       return false;
22
23 }
24
25 void HighScoreState::Draw()
26 {
27
       system("cls");
28
       cout << endl << endl;</pre>
                            - - - - HIGH SCORES - - - - " << endl << ₹
29
       cout << "
          endl;
30
       for (auto i = m_highscore.rbegin(); i != m_highscore.rend(); ++i)
31
32
           cout << "
                                " << *i << endl;
33
34
       }
35
       cout << endl;</pre>
36
37
       cout << endl;</pre>
38
       cout << "Press any key to go back to the main menu." << endl << endl;</pre>
39 }
```

```
...nheritance and Polymorphism\Game\Game Inheritence\Key.h
1 #include "PlaceableActor.h"
 2
 3 class Key : public PlaceableActor
 4 {
 5 public:
        Key(int x, int y, ActorColour colour)
 7
            : PlaceableActor(x, y, colour)
 8
 9
        }
10
        virtual ActorType GetType() override { return ActorType::Key; }
11
        virtual void Draw() override;
12
13 };
```

```
...eritance and Polymorphism\Game\Game Inheritence\Key.cpp
1 #include <iostream>
 2 #include <Windows.h>
 3 #include "Key.h"
 4
 5 void Key::Draw()
 6 {
 7
        HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(console, (int)m_colour);
        std::cout << "+"; // prints coloured key.</pre>
        SetConsoleTextAttribute(console, (int)ActorColour::Regular);
10
11 }
```

```
1 #include "Player.h"
2 #include <string>
 3 #include <vector>
4 using namespace std;
 6 class PlaceableActor;
7
8
9 class Level
10 {
       char* plevel;
11
12
       int height;
13
       int width;
14
15
       vector<PlaceableActor*> m_pActors;
17 public:
18
       Level();
19
       ~Level();
20
       bool LoadLevel(string levelName, int* playerX, int* playerY);
21
       void Draw();
22
23
       PlaceableActor* UpdateActors(int x, int y);
24
25
       bool IsSpace(int x, int y);
26
       bool IsWall(int x, int y);
27
28
       int GetHeight() { return height; }
29
       int GetWidth() { return width; }
       int GetIndex(int x, int y);
30
31
       static constexpr char WAL = (char)219;
32
33
34 private:
35
       bool Convert(int* playerX, int* playerY);
36
37 };
38
```

```
1 #include <Windows.h>
 2 #include "Level.h"
 3 #include <iostream>
4 #include <fstream>
 5 #include "Player.h"
 6 #include "Enemy.h" // derived Placeable Actor Classes
 7 #include "Kev.h"
8 #include "Door.h"
9 #include "Goal.h"
10 #include "Money.h"
11 #include <assert.h>
12
13 using namespace std;
14
15 Level::Level()
16
       : plevel(nullptr)
       , height(0)
17
18
       , width(0)
19 {
20
21 };
22
23 Level::~Level()
24 {
25
       if (plevel != nullptr)
26
            delete[] plevel;
27
            plevel = nullptr;
28
29
       }
30
31
       while (!m pActors.empty())
32
33
            delete m_pActors.back(); // return us the elements at end, then
34
           m_pActors.pop_back(); // continue to delete the remaining vector
              elements.
35
       }
36 };
37
38 bool Level::LoadLevel(string levelName, int* playerX, int* playerY)
39 {
40
       levelName.insert(0, "../");
41
       ifstream levelFile;
42
       levelFile.open(levelName);
       if (!levelFile)
43
44
       {
            cout << "An error has occured." << endl;</pre>
45
46
            return false;
47
       }
48
       else
49
            constexpr int tempSize = 25;
50
51
            char temp[tempSize];
```

```
...itance and Polymorphism\Game\Game Inheritence\Level.cpp
```

```
5
```

```
52
53
            levelFile.getline(temp, tempSize, '\n');
54
            width = atoi(temp); // converts integer into width.
55
56
            levelFile.getline(temp, tempSize, '\n'); // line 83 and line 87
              link.
57
            height = atoi(temp);
58
            plevel = new char[width * height]; // array that we need to
59
              deallocate.
            levelFile.read(plevel, width * height);
60
61
62
            if (playerX != nullptr && playerY != nullptr)
63
64
                bool anyWarnings = Convert(playerX, playerY);
65
                if (anyWarnings)
66
                {
                    cout << "There are some warnings in the level data. see</pre>
67
                      above." << endl;</pre>
68
                    system("pause");
69
                }
70
            }
71
            return true;
72
        }
73 }
74
75 void Level::Draw()
76 {
        HANDLE console = GetStdHandle(STD OUTPUT HANDLE); // temprary variables →
77
          being deleted at the end of draw.
78
        SetConsoleTextAttribute(console, (int)ActorColour::Regular);
79
        //Draw Level
80
        for (int y = 0; y < GetHeight(); ++y)</pre>
81
82
            for (int x = 0; x < GetWidth(); ++x)</pre>
83
84
                int indexToPoint = GetIndex(x, y);
85
                cout << plevel[indexToPoint];</pre>
86
87
            }
            cout << endl;</pre>
88
89
        }
90
        COORD actorCursorPosition; // position the cursor at correct location, x →
91
           and y variables
92
93
        // Draw actors
94
        for (auto actor = m pActors.begin(); actor != m pActors.end(); +
95
          +actor) // going to the beginning and through the end.
96
        {
            if ((*actor)->IsActive()) // if active we want to draw.
97
98
            {
```

```
\dotsitance and Polymorphism\Game\Game Inheritence\Level.cpp
99
                 actorCursorPosition.X = (*actor)->GetXPosition();
100
                 actorCursorPosition.Y = (*actor)->GetYPosition();
101
                 SetConsoleCursorPosition(console, actorCursorPosition); // set
                   position manually to this point.
102
                 (*actor)->Draw(); // draw the actors, tempoary variable in line →
                   93 is now finished and deleted from the stack.
103
             }
104
         }
105 }
106
107 bool Level::IsSpace(int x, int y)
108 {
109
         return plevel[GetIndex(x, y)] == ' ';
110 }
111 bool Level::IsWall(int x, int y)
112 {
113
         return plevel[GetIndex(x, y)] == WAL;
114 }
115
116 bool Level::Convert(int* playerX, int* playerY)
117 {
118
         bool anyWarnings = false;
119
120
         for (int y = 0; y < height; ++y)
121
122
             for (int x = 0; x < width; ++x)
123
124
                 int intIndex = GetIndex(x, y);
125
126
                 switch (plevel[intIndex])
```

127

128

129

130131

132133

134135

136

137

138

139

140

141142

143

144

145146

147

148149

{

{

};

case '+':

case '-':

case '|':

case ' ':

case 'r':

case 'g':

case 'b':

break;

break;

break;

break;

plevel[intIndex] = WAL;

plevel[intIndex] = ' ';

plevel[intIndex] = ' ';

plevel[intIndex] = ' ';

m_pActors.push_back(new Key(x, y, ActorColour::Red));

m_pActors.push_back(new Key(x, y, ActorColour::Green));

m_pActors.push_back(new Key(x, y, ActorColour::Blue));

```
...itance and Polymorphism\Game\Game Inheritence\Level.cpp
150
                     break;
151
                 case 'R':
                     plevel[intIndex] = ' ';
152
                     m_pActors.push_back(new Door(x, y, ActorColour::Red,
153
                       ActorColour::RedSolid));
154
                     break;
155
                 case 'G':
                     plevel[intIndex] = ' ';
156
                     m_pActors.push_back(new Door(x, y, ActorColour::Green,
157
                                                                                     P
                       ActorColour::GreenSolid));
158
                     break;
                 case 'B':
159
160
                     plevel[intIndex] = ' ';
161
                     m_pActors.push_back(new Door(x, y, ActorColour::Blue,
                       ActorColour::BlueSolid));
162
                     break;
                 case 'X':
163
164
                     plevel[intIndex] = ' ';
                     m_pActors.push_back(new Goal(x, y));
165
166
                     break:
                 case '$':
167
                     plevel[intIndex] = ' ';
168
169
                     m_pActors.push_back(new Money(x, y, 1 + rand() % 5));
170
                     break;
171
                 case '@':
172
                 {
                     plevel[intIndex] = ' ';
173
                     if (playerX != nullptr && playerY != nullptr)
174
175
                     {
176
                          *playerX = x;
177
                          *playerY = y;
                     }
178
179
                     break;
                 }
180
181
                 case 'e':
                     m_pActors.push_back(new Enemy(x, y));
182
                     plevel[intIndex] = ' '; // clear level
183
184
                     break;
185
                 case 'h': // horiztonal enemy
186
                     m_pActors.push_back(new Enemy(x, y, 3, 0));
                     plevel[intIndex] = ' ';
187
188
                     break;
                 case 'v': // vertical enemy
189
                     plevel[intIndex] = ' ';
190
191
                     m_pActors.push_back(new Enemy(x, y, 0, 2));
                     plevel[intIndex] = ' ';
192
193
                     break;
194
                 default:
195
                     cout << "Invalid character in file " << plevel[intIndex] << >
196
197
                     anyWarnings = true;
198
                     break;
```

```
...itance and Polymorphism\Game\Game Inheritence\Level.cpp
199
200
                 }
201
            }
202
        }
203
        return anyWarnings;
204 }
205
206 int Level::GetIndex(int x, int y)
207 {
208
        return x + y * width;
209 }
210
211 // Updates all actors and returns a colliding actor is there is one.
212
213 PlaceableActor* Level::UpdateActors(int x, int y ) // pass in x and y of
      player.
214 {
215
216
        PlaceableActor* collidedActor = nullptr;
217
218
        for (auto actor = m_pActors.begin(); actor != m_pActors.end(); ++actor)
219
        {
            (*actor)->Update(); //update all actors
220
221
222
            if (x == (*actor)->GetXPosition() && y == (*actor)->GetYPosition
              ()) // collision occured
223
224
                 assert(collidedActor == nullptr); // if assertion fails, two
```

collidedActor = (*actor); // points to the location of the

points have met.

collision.

return collidedActor;

}

}

225

226

227

228

229 }

```
1 #pragma once
3 #include "StateMachineExampleGame.h"
5 #include "GameState.h"
6 class LoseState :
       public GameState
           StateMachineExampleGame * m_pOwner;
9
10
11
     public:
           LoseState(StateMachineExampleGame* pOwner);
12
13
           ~LoseState() = default;
14
           virtual bool Update(bool processInput = true) override;
15
           virtual void Draw() override;
17 };
18
19
```

```
1 #include "LoseState.h"
 3 #include <iostream>
4 #include <conio.h>
 6 #include "StateMachineExampleGame.h"
7
 8 using namespace std;
10 LoseState::LoseState(StateMachineExampleGame* pOwner)
   : m_pOwner(pOwner)
11
12 {}
13
14 bool LoseState::Update(bool processInput)
15 {
       if (processInput)
16
17
       {
           int input = _getch();
           m_pOwner->LoadScene(StateMachineExampleGame::SceneName::MainMenu);
19
20
       return false;
21
22 }
23
24 void LoseState::Draw()
25 {
       system("cls");
26
       cout << endl << endl;</pre>
27
                                                               " << endl << →
28
       cout << "
                           - - - - GAME OVER - - - -
         endl;
       cout << "
29
                           BETTER LUCK NEXT TIME " << endl << endl;
       cout << "
                            PRESS ANY KEY TO GO BACK TO MAIN MENU
30
         endl << endl;</pre>
31
32 }
```

```
2 #include "GameState.h"
4 class StateMachineExampleGame;
6 class MainMenuState :
7
      public GameState
8 {
      StateMachineExampleGame* m_pOwner;
9
10
11 public:
      MainMenuState(StateMachineExampleGame* pOwner);
12
13
      ~MainMenuState() = default;
14
      virtual bool Update(bool processInput = true) override;
15
16
      virtual void Draw() override;
17
18 };
19
20
```

```
1 #include "MainMenuState.h"
 2
 3 #include <iostream>
 4 #include <conio.h>
 6 #include "StateMachineExampleGame.h"
 7
 8 using namespace std;
10 constexpr int kEscape = 27;
11
12 constexpr char kPlay = '1';
13 constexpr char kHighScore = '2';
14 constexpr char kSettings = '3';
15
16 constexpr char kQuit = '4';
17
18 MainMenuState::MainMenuState(StateMachineExampleGame* pOwner)
19
        : m_pOwner(pOwner)
20 {}
21
22 bool MainMenuState::Update(bool processInput)
23
24
       bool shouldQuit = false;
25
       if (processInput)
26
27
            int input = _getch();
28
           if (input == kEscape || char(input) == kQuit)
29
            {
30
                shouldQuit = true;
31
32
           else if ((char)input == kPlay)
33
34
                m_pOwner->LoadScene
                  (StateMachineExampleGame::SceneName::Gameplay);
35
36
           else if ((char)input == kHighScore)
37
38
                m_pOwner->LoadScene
                  (StateMachineExampleGame::SceneName::Highscore);
39
           }
40
           else if ((char)input == kSettings)
41
42
                m pOwner->LoadScene
                                                                                   ₽
                  (StateMachineExampleGame::SceneName::Settings);
43
            }
44
45
       return shouldQuit;
46 }
47
48 void MainMenuState::Draw()
49 {
50
       system("cls");
```

```
...nd Polymorphism\Game\Game Inheritence\MainMenuState.cpp
```

```
51
        cout << endl << endl;</pre>
                                - - - - MAIN MENU - - - -
                                                                           " << endl << →
52
        cout << "
         endl;
                                      " << kPlay << ". Play " << endl;
        cout << "
53
        cout << "
                                      " << kHighScore << ". Highscore " << endl; " << kSettings << ". Settings " << endl;
54
55
        cout << "
                                      " << kQuit << ". Quit " << endl;
        cout << "
56
57
58 }
```

```
...eritance and Polymorphism\Game\Game Inheritence\Money.h
1 #include "PlaceableActor.h"
 2
 3 class Money : public PlaceableActor
 4 {
 5 public:
 6
        Money(int x, int y, int worth);
 7
        int GetWorth() const { return m_worth; }
 8
        virtual ActorType GetType() override { return ActorType::Money; }
10
        virtual void Draw() override;
11
12
13 private:
14
      int m_worth;
15
16 };
```

```
1 #include "Money.h"
2 #include <iostream>
3
4 Money::Money(int x, int y, int worth)
5 : PlaceableActor(x, y)
6 , m_worth(worth)
7 {
8
9 }
10
11 void Money::Draw()
12 {
13    std::cout << "$";
14 }</pre>
```

```
1 #ifndef PLACEABLEACTOR H
 2 #define PLACEABLEACTOR H
 3 #include "Point.h"
 4
 5 enum class ActorColour
 6 {
 7
       Regular = 7,
       Blue = 9,
 8
 9
       Green = 10,
10
       Red = 12,
11
       GreenSolid = 34,
       RedSolid = 255,
12
13
       BlueSolid = 153
14 };
15
16 enum class ActorType
17 {
18
       Door,
19
       Enemy,
20
       Goal,
21
       Key,
22
       Money,
23
       Player
24 };
25
26 class PlaceableActor
27 {
28 public:
       PlaceableActor(int x, int y, ActorColour colour = ActorColour::Regular);
29
30
       virtual ~PlaceableActor();
31
32
       int GetXPosition();
33
       int GetYPosition();
34
       int* GetXPositionPointer();
35
       int* GetYPositionPointer();
       void SetXYPosition(int x, int y);
36
37
38
       ActorColour GetColour() { return m_colour; }
39
40
       void Remove() { m_IsActive = false; }
       bool IsActive() { return m_IsActive; }
41
42
       void Place(int x, int y);
43
44
       virtual ActorType GetType() = 0;
45
       virtual void Draw() = 0;
       virtual void Update() // some placeable actors will not need to update
46
          themselves
47
        {
48
49
       }
50
51 protected:
52
       Point* m_pPosition;
```

```
...and Polymorphism\Game\Game Inheritence\PlaceableActor.h
53 bool m IsActive:
```

```
54 ActorColour m_colour;
55
56 };
57
58 #endif
```

```
1 #include "PlaceableActor.h"
 2
 3 PlaceableActor::PlaceableActor(int x, int y, ActorColour colour)
       : m_pPosition(new Point(x, y))
 4
 5
       , m_IsActive(true),
       m_colour(colour)
 6
 7 {
 8
 9
   }
10
11 PlaceableActor::~PlaceableActor()
12 {
13
       delete m_pPosition;
       m pPosition = nullptr;
14
15 }
17 int PlaceableActor::GetXPosition()
19
       return m_pPosition->x;
20 }
21
22 int PlaceableActor::GetYPosition()
23 {
24
       return m_pPosition->y;
25 }
26
27 int* PlaceableActor::GetXPositionPointer()
28 {
29
       return &(m_pPosition->x);
30 }
31
32 int* PlaceableActor::GetYPositionPointer()
34
       return &(m_pPosition->y);
35 }
36
37 void PlaceableActor::SetXYPosition(int x, int y)
38 {
39
       m_pPosition->x = x;
40
       m_pPosition->y = y;
41 }
42
43 void PlaceableActor::Place(int x, int y)
44 {
45
       m_pPosition->x = x;
46
       m_pPosition->y = y;
47
       m_IsActive = true;
48 }
```

```
1 #ifndef _PLAYER_H_
 2 #define _PLAYER_H_
 3
4 #include "PlaceableActor.h"
 6 class Key; // you can only forward declare pointer types, specific items
 7
 8 class Player : public PlaceableActor
9 {
10
11 public:
12
       Player();
13
       bool HasKey();
14
15
       bool HasKey(ActorColour colour);
16
       void PickUpKey(Key* key);
17
       void UseKey();
18
       void DropKey();
19
       Key* GetKey() { return m_pCurrentKey; }
20
       // nothing in the class is using key in a functions
21
22
23
       void AddMoney(int money) { m_money += money; }
24
       int GetMoney() { return m_money; }
25
       int GetLive() { return m_lives; }
26
27
       void DecrementLives() { m_lives--; }
28
       virtual ActorType GetType() override { return ActorType::Player; }
29
30
       virtual void Draw() override;
31
32 private:
33
       Key* m_pCurrentKey;
34
       int m_money;
35
       int m_lives;
36
37 };
38
39 #endif // !_PLAYER_H_
40
```

```
1 #include "Player.h"
 2 #include "Key.h" // using key in a function
 3 #include <iostream>
 5 using namespace std;
 6
 7 constexpr int kStartNumberOfLives = 1;
 8
9 Player::Player()
10
       : PlaceableActor(0,0)
11
       , m_pCurrentKey(nullptr)
12
       , m_money(0)
13
       , m_lives(kStartNumberOfLives)
14 {
15 };
16
17 bool Player::HasKey()
19
       return m_pCurrentKey != nullptr;
20 }
21
22 bool Player::HasKey(ActorColour colour)
23 {
24
       return HasKey() && m_pCurrentKey->GetColour() == colour;
25 }
26
27 void Player::PickUpKey(Key* key)
28 {
29
       m_pCurrentKey = key;
30 }
31
32 void Player::UseKey()
33 {
34
       m_pCurrentKey->Remove();
35
       m_pCurrentKey = nullptr;
36 }
37
38 void Player::DropKey()
39 {
40
       if (m_pCurrentKey)
41
42
           m_pCurrentKey->Place(m_pPosition->x, m_pPosition->y);
43
           m_pCurrentKey = nullptr;
44
       }
45 }
46
47 void Player::Draw()
48 {
49
       cout << "@";
50 }
```

```
2 struct Point
3 {
4
       int x;
5
       int y;
6
       Point()
7
8
           : x(0)
9
           , y(0)
10
       {
11
12
       }
13
14
       Point(int x, int y)
15
16
           this->x = x;
17
           this->y = y;
18
       }
19
20 };
21
22
23
24
```

```
1 #include <iostream>
2 #include <conio.h>
3 #include <Windows.h>
4 #include <fStream>
6 #include "StateMachineExampleGame.h"
7 #include "AudioManager.h"
8 #include "Game.h"
9 using namespace std;
10
11 int main() {
12
13
       Game myGame;
14
       StateMachineExampleGame gameStateMachine(&myGame);
15
16
       myGame.Initialise(&gameStateMachine);
17
       myGame.RunGameLoop();
19
       myGame.Deinitialise();
20
       AudioManager::destroyInstance();
21
22
23
       return 0;
24
25 }
26
27
```

```
...e and Polymorphism\Game\Game Inheritence\SettingState.h
1 #pragma once
 2 #include "GameState.h"
 4 class StateMachineExampleGame;
 6 class SettingState :
 7
       public GameState
 8 {
       StateMachineExampleGame* m_pOwner;
 9
10
11 public:
       SettingState(StateMachineExampleGame* pOwner);
12
13
       ~SettingState() = default;
14
       virtual bool Update(bool processInput = true) override;
15
       virtual void Draw() override;
16
17
18 };
19
20
```

```
1 #include "SettingState.h"
 2
 3 #include <iostream>
 4 #include <conio.h>
 6 #include "StateMachineExampleGame.h"
 7 #include "AudioManager.h"
 8
 9
   using namespace std;
10
11 constexpr int kEscape = 27;
12
13 constexpr char kSound = '1';
14 constexpr char kMainMenu = '2';
15
16    SettingState::SettingState(StateMachineExampleGame* pOwner)
17
        : m_pOwner(pOwner)
18 {}
19
20 bool SettingState::Update(bool processInput)
21 {
       if (processInput)
22
23
24
            int input = _getch();
25
            if (input == kEscape || char(input) == kMainMenu)
26
27
                m pOwner->LoadScene
                  (StateMachineExampleGame::SceneName::MainMenu);
28
            }
            else if ((char)input == kSound)
29
30
            {
                AudioManager::GetInstance()->ToggleSound();
31
                AudioManager::GetInstance()->moneySound();
32
33
            }
34
       }
       return false;
35
36 }
37
38 void SettingState::Draw()
39 {
       system("cls");
40
41
        cout << endl << endl;</pre>
42
       cout << "
                              - - - - SETTINGS - - - -
                                                                   " << endl;
                                  " << kSound << ". Play " << endl;
       cout << "
43
                                  " << "Toggle Sound: ";
       cout << "
44
       if (AudioManager::GetInstance()->IsSoundOn())
45
46
47
            cout << "ON" << endl;</pre>
       }
48
49
       else
50
       {
            cout << "OFF" << endl;</pre>
51
52
       }
```

54 55 }

```
1 #include "StateMachineExampleGame.h"
 2
 3 #include "MainMenuState.h"
 4 #include "GameplayState.h"
 5 #include "SettingState.h"
 6 #include "HighScoreState.h"
 7 #include "WinState.h"
 8 #include "LoseState.h"
10 #include "Game.h"
11
12 StateMachineExampleGame::StateMachineExampleGame(Game* pOwner)
13
        : m_pOwner(pOwner)
       , m pCurrentState(nullptr)
14
15
       , m_pNewState(nullptr)
16 {};
17
18 bool StateMachineExampleGame::Init()
19 {
20
       LoadScene(SceneName::MainMenu);
21
       return true;
22 }
23
24 bool StateMachineExampleGame::UpdateCurrentState(bool processInput)
25 {
26
       bool done = false;
27
       if (m pNewState != nullptr)
28
29
           ChangeState(m_pNewState);
30
           m_pNewState = nullptr;
31
       }
32
33
       if (m_pCurrentState != nullptr)
34
35
           done = m_pCurrentState->Update(processInput);
36
37
       return done;
38 }
39
40 void StateMachineExampleGame::DrawCurrentState()
41 {
42
       if (m_pCurrentState)
43
       {
44
           m_pCurrentState->Draw();
45
46 }
47
48 void StateMachineExampleGame::ChangeState(GameState* pNewState)
49
50
       if (m_pCurrentState)
51
       {
52
           m_pCurrentState->Exit();
53
```

```
54
55
       delete m_pCurrentState;
56
       m_pCurrentState = pNewState;
57
       pNewState->Enter();
58 }
59
60 void StateMachineExampleGame::CleanUp()
61 {
62
       if (m_pCurrentState)
63
       {
            m_pCurrentState->Exit();
64
65
            delete m_pCurrentState;
66
       }
67 }
68
69 void StateMachineExampleGame::LoadScene(SceneName scene)
70 {
71
       switch (scene)
72
73
       case SceneName::MainMenu:
74
            m_pNewState = new MainMenuState(this);
75
            break;
76
        case SceneName::Gameplay:
77
            m_pNewState = new GameplayState(this);
78
            break;
79
       case SceneName::Settings:
            m_pNewState = new SettingState(this);
80
81
            break;
82
        case SceneName::Highscore:
            m_pNewState = new HighScoreState(this);
83
84
            break;
       case SceneName::Win:
85
86
            m_pNewState = new WinState(this);
87
            break;
88
       case SceneName::Lose:
89
            m_pNewState = new LoseState(this);
90
            break;
91
       default:
92
            break;
93
       }
94 }
```

```
1 #pragma once
 2 #include "GameStateMachine.h"
 3
 4 class Game;
 5 class GameState;
 7 class StateMachineExampleGame :
 8
       public GameStateMachine
 9 {
10 public:
       enum class SceneName
11
12
13
            None,
14
            MainMenu,
15
            Gameplay,
16
            Settings,
17
            Highscore,
18
            Lose,
19
           Win
20
       };
21
22 private:
23
       Game* m_pOwner;
24
25
       GameState* m pCurrentState;
       GameState* m_pNewState;
26
27
28 public:
       StateMachineExampleGame(Game* pOwner);
29
30
       virtual bool Init() override;
31
32
       virtual bool UpdateCurrentState(bool processInput = true) override;
       virtual void DrawCurrentState() override;
       virtual void ChangeState(GameState* pNewState) override;
34
35
       virtual void CleanUp() override;
       void LoadScene(SceneName scene);
36
37
38 };
39
40
```

```
1 #pragma once
 2
 3 #include <iostream>
 4 #include <set>
 5 #include <string>
 6 #include <fstream>
 7 #include <iterator>
 8
 9
   using namespace std;
10
11 class Utility
12 {
13 public:
       static set<int> WriteHighScore(int score)
14
15
16
            // see if file exists and read values
17
           string fileName = "highscores.txt";
19
           ifstream highScoreFile(fileName);
20
           istream iterator<int> start(highScoreFile), end;
21
            set<int> highscores(start, end);
22
           highScoreFile.close();
23
24
           // if its empty, populate and save it.
25
26
           if (highscores.size() == 0)
27
            {
                highscores.insert(100);
28
29
                highscores.insert(50);
30
                highscores.insert(20);
31
                highscores.insert(10);
32
                highscores.insert(5);
33
34
                ofstream outFile(fileName);
35
                ostream_iterator<int> output_iterator(outFile, "\n");
                copy(highscores.begin(), highscores.end(), output_iterator);
36
37
                outFile.close();
38
           }
39
40
           // write score
41
42
           highscores.insert(score);
43
44
           // remove lowest score
45
46
           highscores.erase(highscores.begin());
47
48
           // write the highscores.
49
           ofstream outFile(fileName);
50
           ostream_iterator<int> output_iterator(outFile, "\n");
51
            copy(highscores.begin(), highscores.end(), output_iterator);
           outFile.close();
52
53
```

```
...itance and Polymorphism\Game\Game Inheritence\Utility.h

return highscores:
```

```
55 }
56
57
58 };
```

```
2 #include "GameState.h"
3
4 #include "StateMachineExampleGame.h"
6 class WinState :
7
      public GameState
8 {
      StateMachineExampleGame* m_pOwner;
9
10
11 public:
      WinState(StateMachineExampleGame* pOwner);
12
13
      ~WinState() = default;
14
      virtual bool Update(bool processInput = true) override;
15
      virtual void Draw() override;
16
17
18 };
19
20
```

```
1 #include "WinState.h"
 2
 3 #include <iostream>
4 #include <conio.h>
 6 #include "StateMachineExampleGame.h"
7
8 using namespace std;
10 WinState::WinState(StateMachineExampleGame* pOwner)
    : m_pOwner(pOwner)
11
12 {}
13
14 bool WinState::Update(bool processInput)
15 {
       if (processInput)
16
17
       {
           int input = _getch();
           m_pOwner->LoadScene(StateMachineExampleGame::SceneName::MainMenu);
19
20
       return false;
21
22 }
23
24 void WinState::Draw()
25 {
       system("cls");
26
27
       cout << endl << endl;</pre>
                            - - - - WELL DONE - - - -
28
       cout << "
                                                                " << endl << >
         endl;
29
       cout << "
                           YOU BEAT THE GAME. " << endl << endl;
       cout << "
                                                         " << endl << endl;</pre>
30
31
32 }
33
```